Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 24, 2017

J. Kunze
California Digital Library
J. Littman
George Washington University Libraries
L. Madden
Library of Congress
E. Summers
University of Maryland
A. Boyko

B. Vargas
October 21, 2016

The BagIt File Packaging Format (V0.97)
draft-kunze-bagit-14

Abstract

This document specifies BagIt, a hierarchical file packaging format for storage and transfer of arbitrary digital content. A "bag" has just enough structure to enclose descriptive "tags" and a "payload" but does not require knowledge of the payload's internal semantics. This BagIt format should be suitable for disk-based or network-based storage and transfer. BagIt is widely used in the practice of digital preservation.

Table of Contents

1. Introduction

1.1. Purpose

BagIt is a hierarchical file packaging format designed to support disk-based or network-based storage and transfer of arbitrary digital content. A bag consists of a "payload" and "tags". The content of the payload is the custodial focus of the bag and is treated as semantically opaque. The "tags" are metadata files intended to facilitate and document the storage and transfer of the bag. The name, BagIt, is inspired by the "enclose and deposit" method [ENCDEP], sometimes referred to as "bag it and tag it".

BagIt is widely used for preserving digital assets originating from a different domains. Organizations involved in digital preservation with BagIt include the Library of Congress, Dryad Data Repository, NSF DataONE, and the Rockefeller Archive Center. Software implementations have been written in Python, Ruby, Java, Perl, and PHP. It is also used in the libraries of many universities, such as Cornell, Purdue, Stanford, Ghent University, New York University, and the University of California.

Implementors of BagIt tools should consider interoperability between different platforms, operating systems, toolsets, and languages. Differences in path separators, newline characters, reserved file names, and maximum path lengths are all possible barriers to moving bags between different systems. Discussion of these issues may be found in the Interoperability section of this document.

1.2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.3. Terminology

This specification uses a number of terms to describe BagIt, some of which are in common use, some of which are newly defined by this specification, and others which may have meanings obvious only to those in the community from which this spec arose. Terms defined in this section are intended to clarify any ambiguity.

bag A set of opaque data contained within the structure defined by this specification.

bag declaration The tag file required to be in all bags conforming to this specification. Contains tags necessary for bootstrapping the reading and processing of the rest of a bag. See Section 2.1.1.

bag checksum algorithm A reference to a cryptographic checksum algorithm, such as SHA1 or SHA256, with its name normalized for use in a manifest or tag manifest file name. See Section 2.4.

complete A bag which comprises all elements required by this specification, with all files listed in all payload and tag manifests present, all payload files present listed in at least one manifest. See Section 3.

payload The data encapsulated by the bag. The contents of the payload are opaque to this specification, and are always considered as a set of octet streams. See Section 2.1.2.

serialized bag A bag that has been serialized into a single, monolithic file. See Section 4.

tag directory A directory that contains one or more tag files.

tag file A file that contains metadata intended to facilitate and document the storage and transfer of the bag.

valid A complete bag wherein every checksum in every payload manifest and tag manifest can be successfully verified against the corresponding payload file. See Section 2.1.2.

2. Structure

A bag consists of a base directory containing (1) a set of required and optional tag files; (2) a sub-directory named "data", called the payload directory; and (3) a set of optional tag directories. The payload files in the payload directory are an arbitrary file hierarchy (see Section 2.1.2). The tag files in the base directory consist of one or more files named "manifest-_algorithm_.txt" (see Section 2.1.3), a file named "bagit.txt" (see Section 2.1.1), and zero or more additional tag files (see Section 2.2). The tag files in the optional tag directories are arbitrary file hierarchies and the tag directories MAY have any name that is not reserved for a file or directory in this specification.

The base directory MAY have any name.

```
<base directory>/
|   bagit.txt
|   manifest-<algorithm>.txt
|   [optional additional tag files]
\--- data/
      |   [payload files]
\--- [optional tag directories]/
      |   [optional tag files]
```

2.1. Required Elements

2.1.1. Bag Declaration: bagit.txt

The "bagit.txt" tag file MUST consist of exactly two lines:

```
BagIt-Version: M.N
Tag-File-Character-Encoding: UTF-8
```

where M.N identifies the BagIt major (M) and minor (N) version numbers, and UTF-8 identifies the character set encoding of tag files. The bag declaration MUST be encoded in UTF-8, and MUST NOT contain a byte-order mark (BOM). [RFC3629]

The appropriate version for a bag that conforms to this version of the specification is "0.97".

2.1.2. Payload Directory: data/

The base directory MUST contain a sub-directory named "data", called the payload directory.

The payload directory contains the custodial content within the bag. The files under the payload directory are called payload files, or the payload. The payload is treated as octet streams for all purposes relating to this specification, and is not otherwise prescribed.

2.1.3. Payload Manifest: manifest-<alg>.txt

A payload manifest is a tag file that lists payload files and checksums for those payload files generated using a particular bag checksum algorithm. Every bag MUST contain at least one payload manifest file. A payload manifest file MUST have a name of the form manifest-_algorithm_.txt, where _algorithm_ is a string specifying the bag checksum algorithm used in that manifest, such as:

```
manifest-sha256.txt
manifest-sha1.txt
```

A bag MUST NOT contain more than one payload manifest for a particular bag checksum algorithm.

Each line of a payload manifest file MUST be of the form:

CHECKSUM FILENAME

where FILENAME is the pathname of a file relative to the base directory and CHECKSUM is a hex-encoded checksum calculated according to _algorithm_ over every octet in the file. The hex-encoded checksum MAY use uppercase and/or lowercase letters. The slash character ('/') MUST be used as a path separator in FILENAME. One or more linear whitespace characters (spaces or tabs) MUST separate CHECKSUM from FILENAME. An asterisk ('*') MAY preceed FILENAME for interoperability on some platforms (see Section 7.2.1). There is no limitation on the length of a pathname. The payload manifest MUST NOT reference files outside the payload directory.

Payload manifests only include the pathnames of files. Because of this, a payload manifest cannot reference empty directories. To account for an empty directory, a bag creator may wish to include at least one file in that directory; it suffices, for example, to include a zero-length file named ".keep".

2.2. Optional Elements

2.2.1. Tag Manifest: tagmanifest-<alg>.txt

A tag manifest is a tag file that lists other tag files and checksums for those tag files generated using a particular bag checksum algorithm. A bag MAY contain one or more tag manifests. A tag manifest file MUST have a name of the form "tagmanifest-_algorithm_.txt", where _algorithm_ is a string specifying the bag checksum algorithm used in that manifest, such as:

tagmanifest-sha256.txt
tagmanifest-sha1.txt

A tag manifest file has the same form as the payload file manifest file described in Section 2.1.3, but MUST NOT list any payload files. As a result, no FILENAME listed in a tag manifest begins "data/".

2.2.2. Bag Metadata: bag-info.txt

The "bag-info.txt" file is a tag file that contains metadata elements describing the bag and the payload. The metadata elements contained in the "bag-info.txt" file are intended primarily for human readability. All metadata elements are optional and MAY be repeated.

Implementations SHOULD assume that the ordering is significant and provide access to the metadata elements in the order they are given in the "bag-info.txt" file.

A metadata element MUST consist of a label, a colon, and a value, each separated by optional whitespace. The label MUST start in column 1. It is RECOMMENDED that lines not exceed 79 characters in length. Long values may be continued onto the next line by inserting a newline (LF), a carriage return (CR), or carriage return plus newline (CRLF) and indenting the next line with linear white space (spaces or tabs).

Reserved metadata element names are case-insensitive and defined as follows.

Source-Organization Organization transferring the content.

Organization-Address Mailing address of the organization.

Contact-Name Person at the source organization who is responsible for the content transfer.

Contact-Phone International format telephone number of person or position responsible.

Contact-Email Fully qualified email address of person or position responsible.

External-Description A brief explanation of the contents and provenance.

Bagging-Date Date (YYYY-MM-DD) that the content was prepared for delivery.

External-Identifier A sender-supplied identifier for the bag.

Bag-Size Size or approximate size of the bag being transferred, followed by an abbreviation such as MB (megabytes), GB, or TB; for example, 42600 MB, 42.6 GB, or .043 TB. Compared to Payload-Oxum (described next), Bag-Size is intended for human consumption.

Payload-Oxum The "octetstream sum" of the payload, namely, a two-part number of the form "OctetCount.StreamCount", where OctetCount is the total number of octets (8-bit bytes) across all payload file content and StreamCount is the total number of payload files. Payload-Oxum should be included in "bag-info.txt" if at all possible. Compared to Bag-Size (above), Payload-Oxum is intended for machine consumption.

Bag-Group-Identifier A sender-supplied identifier for the set, if any, of bags to which it logically belongs. If this identifier is recognizable as belonging to a globally unique scheme, the receiver should make an effort to honor reference to it.

Bag-Count Two numbers separated by "of", in particular, "N of T", where T is the total number of bags in a group of bags and N is the ordinal number within the group; if T is not known, specify it as "?" (question mark). Examples: 1 of 2, 4 of 4, 3 of ?, 89 of 145.

Internal-Sender-Identifier An alternate sender-specific identifier for the content and/or bag.

Internal-Sender-Description A sender-local prose description of the contents of the bag.

In addition to these metadata elements, other arbitrary metadata elements may also be present.

Here is an example "bag-info.txt" file.

```
Source-Organization: Spengler University
Organization-Address: 1400 Elm St., Cupertino, California, 95014
Contact-Name: Edna Janssen
Contact-Phone: +1 408-555-1212
Contact-Email: ej@spengler.edu
External-Description: Uncompressed greyscale TIFF images from the
     Yoshimuri papers colle...
Bagging-Date: 2008-01-15
External-Identifier: spengler_yoshimuri_001
Bag-Size: 260 GB
Payload-Oxum: 279164409832.1198
Bag-Group-Identifier: spengler_yoshimuri
Bag-Count: 1 of 15
Internal-Sender-Identifier: /storage/images/yoshimuri
Internal-Sender-Description: Uncompressed greyscale TIFFs created
     from microfilm and are...
```

2.2.3. Fetch File: fetch.txt

For reasons of efficiency, a bag MAY be sent with a list of files to be fetched and added to the payload before it can meaningfully be checked for completeness. An OPTIONAL tag file named "fetch.txt" contains such a list. Each line of "fetch.txt" has the form

URL LENGTH FILENAME

where URL [RFC3986] identifies the file to be fetched, LENGTH is the number of octets in the file (or "-", to leave it unspecified), and FILENAME identifies the corresponding payload file, relative to the base directory. The slash character ('/') MUST be used as a path separator in FILENAME. If FILENAME begins with a slash character, the destination MUST still be treated as relative to the bag base directory. One or more linear whitespace characters (spaces or tabs) MUST separate these three values, and any such characters in the URL MUST be percent-encoded. There is no limitation on the length of any of the fields in the "fetch.txt".

The "fetch.txt" file allows a bag to be transmitted with "holes" in it, which can be practical for several reasons. For example, it obviates the need for the sender to stage a large serialized copy of the content while the bag is transferred to the receiver. Also, this method allows a sender to construct a bag from components that are either a subset of logically related components (e.g., the localized logical object could be much larger than what is intended for export) or assembled from logically distributed sources (e.g., the object components for export are not stored locally under one filesystem tree).

2.2.4. Other Tag Files

A bag MAY contain other tag files that are not defined by this specification. Implementations SHOULD ignore the content of any unexpected tag files, except when they are listed in a tag manifest. When unexpected tag files are listed in a tag manifest, implementations MUST only treat the content of those tag files as octet streams for the purpose of checksum verification.

2.3. Text Tag File Format

All tag files specifically described in this specification MUST adhere to the text tag file format described below. Other tag files MAY adhere to the text tag file format described below.

Text tag files are line-oriented, and each line MUST be terminated by a newline (LF), a carriage return (CR), or carriage return plus newline (CRLF). Text tag files MUST end in the extension ".txt".

In all text tag files except for the bag declaration file, text MUST be encoded in the character encoding specified in the "bagit.txt" bag declaration file. Text tag files except for the bag declaration file MAY include a byte-order mark (BOM) only if the specified encoding requires it for proper decoding. (Note that UTF-8 does not.)

As specified in Section 2.1.1, the bag declaration file must be encoded in UTF-8 and must not include a byte-order mark.

2.4. Bag Checksum Algorithms

The payload manifest and tag manifests assert integrity of the payload and tags in a bag using checksum algorithms. The operation of those algorithms, and the formatting of their output within a manifest file, are generally beyond the scope of this specification, except that the output format MUST be able to fit in the manifest format specified in Section 2.1.3.

The name of the checksum algorithm MUST be normalized for use in the manifest's filename by lowercasing the common name of the algorithm and removing all non-alphanumeric characters.

Implementors of tools that create and validate bags SHOULD support at least two widely implemented checksum algorithms: "md5" [RFC1321] and "sha1" [RFC3174]. The authors recognize that, compared with newer algorithms [RFC6234], these two algorithms now have well-known vulnerabilities that render them inadequate for applications requiring secure change detection.

3. Complete, Incomplete, and Valid bags

A _complete_ bag MUST have the following attributes:

1. Every required element MUST be present (Section 2.1).

2. Every file in every payload manifest MUST be present.

3. Every file in every tag manifest MUST be present. Tag files not listed in a tag manifest MAY be present.

4. Every payload file MUST be listed in at least one manifest. Payload files MAY be listed in more than one payload manifest.

5. Every element present MUST comply with this specification.

A bag is _incomplete_ when it exhibits any of the following exceptions to the attributes of a complete bag:

1. One or more files in any payload manifest are absent.

2. One or more files in any tag manifest are absent.

3. A fetch.txt is present. Any files listed in any payload manifest or any tag manifest which are absent MUST be listed in the fetch.txt.

A _valid_ bag must have the following attributes:

1. The bag MUST be complete.

2. Every CHECKSUM in every payload manifest and tag manifest can be sucessfully verified against the contents of its corresponding FILENAME.

If a bag is neither valid, complete, nor incomplete, it is _invalid_. Definitions for the various ways a bag may be invalid are not covered by this specification.

Tag files that do not appear in a tag manifest can be modified, added to, or removed from a bag without impacting the completeness or validity of the bag.

4. Serialization

In some scenarios, it might be convenient to serialize the bag's filesystem hierarchy (i.e., the base directory) into a single-file archive format such as TAR or ZIP (the serialization) and then later deserialize the serialization to recreate the filesystem hierarchy. Several rules govern the serialization of a bag and apply equally to all types of archive files:

1. The top-level directory of a serialization MUST contain only one bag.

2. The serialization SHOULD have the same name as the bag's base directory, but MUST have an extension added to identify the format. For example, the receiver of "mybag.tar.gz" expects the corresponding base directory to be created as "mybag".

3. A bag MUST NOT be serialized from within its base directory, but from the parent of the base directory (where the base directory appears as an entry). Thus, after a bag is deserialized in an empty directory, a listing of that directory shows exactly one entry. For example, deserializing "mybag.zip" in an empty directory causes the creation of the base directory "mybag" and, beneath "mybag", the creation of all payload and tag files.

4. The deserialization of a bag MUST produce a single base directory bag with the top-level structure as described in this specification without requiring any additional un-archiving step.

For example, after one un-archiving step it would be an error for the "data/" directory to appear as "data.tar.gz". TAR and ZIP files may appear inside the payload beneath the "data/" directory, where they would be treated as any other payload file.

When serializing a bag, care must be taken to ensure that the archive format's restrictions on file naming, such as allowable characters, length, or character encoding, will support the requirements of the systems on which it will be used. See Section 7.2.

5. Examples

5.1. Example of a basic bag

This is the layout of a basic bag containing an image and a companion OCR file. Lines of file content are shown in parentheses beneath the file name. For brevity, examples use the md5 checksum algorithm.

```
myfirstbag/
|
|   manifest-md5.txt
|    (49afbd86a1ca9f34b677a3f09655eae9 data/27613-h/images/q172.png)
|    (408ad21d50cef31da4df6d9ed81b01a7 data/27613-h/images/q172.txt)
|
|   bagit.txt
|    (BagIt-version: 0.96                                          )
|    (Tag-File-Character-Encoding: UTF-8                           )
|
\--- data/
     |
     |   27613-h/images/q172.png
     |    (... image bytes ...                                     )
     |
     |   27613-h/images/q172.txt
     |    (... OCR text ...                                        )
     ....
```

5.2. Another example bag

The following example bag contains content from a web crawler. As before, lines of file content are shown in parentheses beneath the file name, with long lines continued indented on subsequent lines. This bag is not complete until every component listed in the "fetch.txt" file is retrieved.

```
 mysecondbag/
 |
 |   manifest-md5.txt
 |    (93c53193ef96732c76e00b3fdd8f9dd3 data/Collection Overview.txt   )
 |    (e9c5753d65b1ef5aeb281c0bb880c6c8 data/Seed List.txt             )
 |    (61c96810788283dc7be157b340e4eff4 data/gov-20060601-050019.arc.gz)
 |    (55c7c80c6635d5a4c8fe76a940bf353e data/gov-20060601-100002.arc.gz)
 |
 |   fetch.txt
 |    (http://WB20.Stanford.Edu/gov-06-2006/gov-20060601-050019.arc.gz
 |        26583985 data/gov-20060601-050019.arc.gz                     )
 |    (http://WB20.Stanford.Edu/gov-06-2006/gov-20060601-100002.arc.gz
 |        99509720 data/gov-20060601-100002.arc.gz                     )
 |    ( ..............................................................)
 |
 |   bag-info.txt
 |    (Source-organization: California Digital Library                 )
 |    (Organization-address: 415 20th St, 4th Floor, Oakland, CA  94612)
 |    (Contact-name: A. E. Newman                                      )
 |    (Contact-phone: +1 510-555-1234                                  )
 |    (Contact-email: alfred@ucop.edu                                  )
 |    (External-Description: The collection "Local Davis Flood Control )
 |      Collection" includes captured California State and local       )
 |      websites containing information on flood control resources for )
 |      the Davis and Sacramento area.  Sites were captured by UC Davis)
 |      curator Wrigley Spyder using the Web Archiving Service in      )
 |      February 2007 and October 2007.                                )
 |    (Bag-date: 2008.04.15                                            )
 |    (External-identifier: ark:/13030/fk4jm2bcp                       )
 |    (Bag-size: about 22Gb                                            )
 |    (Payload-Oxum: 21836794142.831                                   )
 |    (Internal-sender-identifier: UCDL                                )
 |    (Internal-sender-description: UC Davis Libraries                 )
 |
 |   bagit.txt
 |    (BagIt-version: 0.96                                             )
 |    (Tag-File-Character-Encoding: UTF-8                              )
 |
 \--- data/
      |
      |   Collection Overview.txt
      |    (... narrative description ...                             )
      |
      |   Seed List.txt
      |    (... list of crawler starting point URLs ...               )
      ....
```

6. Security Considerations

6.1. Special directory characters

The paths specified in the payload manifest, tag manifest, and "fetch.txt" file do not prohibit special directory characters which might be significant on implementing systems. Implementors SHOULD take care that files outside the bag directory structure are not accessed when reading or writing files based on paths specified in a bag.

For example, path characters such as ".." or "~" in a maliciously crafted "fetch.txt" file might cause a naive implementation to overwrite critical system files.

6.2. Control of URLs in fetch.txt

Implementors of tools that complete bags by retrieving URLs listed in a "fetch.txt" file need to be aware that some of those URLs may point to hosts, intentionally or unintentionally, that are not under control of the bag's sender. Checksums are intended as a reasonable guarantee against corruption during transit, not a strong cryptographic protection against intentional spoofing.

6.3. File sizes in fetch.txt

The size of files, as optionally reported in the "fetch.txt" file, cannot be guaranteed to match the actual file size to be downloaded. Implementors SHOULD take care to appropriately handle cases where the actual file size does not match the file size reported in the fetch.txt. Implementors SHOULD NOT use the file size in the "fetch.txt" file for critical resource allocation, such as buffer sizing or storage requisitioning.

7. Practical Considerations (non-normative)

7.1. Disk and network transfer

When creating a bag on physical media (such as hard disk, CD-ROM, or DVD) for transfer to another organization, the sender should select and format the media in a manner compatible with both the content requirements (e.g., file names and sizes) and the receiver's technical infrastructure. If the receiver's infrastructure is not known or the media needs to be compatible with a range of potential receivers, consideration should be given to portability and common usage. For example, a "lowest common denominator" for some potential receivers could be USB disk drives formatted with the FAT32 filesystem.

Although overall bag size is unlimited in principle, network-based transfers might involve constraints on the amount of bag data that a receiver can receive at one time. It might be practical to split a large bag into several smaller bags.

Transmitting a whole bag in serialized form as a single file will tend to be the most straightforward mode of transfer. When throughput is a priority, use of "fetch.txt" lends itself to an easy, application-level parallelism in which the list of URL-addressed items to fetch is divided among multiple processes. The mechanics of sending and receiving bags over networks is otherwise out of scope of the present document and might be facilitated by protocols such as [GRABIT] and [SWORD].

7.2. Interoperability

This section is not part of the BagIt specification. It describes some practical considerations for bag creators and receivers circa 2010.

7.2.1. Checksum tools

Some cautions regarding bag interchange arise in regard to the commonly available checksum tools distributed with the GNU Coreutils package (md5sum, sha1sum, sha256sum, etc.), collectively referred to here as "sha256sum". First, sha256sum can be run in binary or text mode; text mode sometimes normalizes line-endings. While these modes appear to produce the same checksums under Unix-like systems, they can produce different checksums under Windows. When using sha256sum, it might be safest to run it in binary mode, with one caveat: a side-effect of binary mode is that sha256sum requires a space and an asterisk ('*'), compared to two spaces in text mode, between the CHECKSUM and FILENAME in its manifest format.

Due to the widespread use of sha256sum (and its relatives), it is not unexpected for bag receivers to see manifests in which CHECKSUM and FILENAME are separated by a space followed by an asterisk. Implementors creating or processing bags with sha256sum should be aware of these subtle differences, and ensure compliance with the manifest specification in this document. Implementors creating and processing bags with other tools might wish to be tolerant of asterisks found in the manifests.

A final note about sha256sum-generated manifests is that for a FILENAME containing a backslash ('\'), the manifest line will have a backslash inserted in front of the CHECKSUM and, under Windows, the backslashes inside FILENAME might be doubled.

7.2.2. Windows and Unix file naming

As specified above, only the Unix-based path separator ('/') may be used inside filenames listed in BagIt manifests and "fetch.txt" files. When bags are exchanged between Windows and Unix platforms, care should be taken to translate the path separator as needed. Receivers of bags on physical media should be prepared for filesystems created under either Windows or Unix. Besides the fundamental difference between path separators ('\' and '/'), generally, Windows filesystems have more limitations than Unix filesystems. Windows path names have a maximum of 255 characters, and none of these characters may be used in a path component:

```
< > : " / | ? *
```

Windows also reserves the following names: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. See [MSFNAM] for more information.

8. Acknowledgements

BagIt owes much to many thoughtful contributers and reviewers, including Stephen Abrams, Mike Ashenfelder, Dan Chudnov, Brad Hards, Scott Fisher, Keith Johnson, Erik Hetzner, Leslie Johnston, David Loy, Mark Phillips, Tracy Seneca, Brian Tingle, Adam Turoff, and Jim Tuttle.

9. IANA Considerations

This draft does not request any action from IANA.

10. References

10.1. Normative References

[MSFNAM] Microsoft, , "Naming a File", 2008, <http://msdn2.microsoft.com/en-us/library/aa365247.aspx>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <http://www.rfc-editor.org/info/rfc1321>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <http://www.rfc-editor.org/info/rfc3174>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

10.2. Informative References

[ENCDEP] Tabata, K., "A Collaboration Model between Archival Systems to Enhance the Reliability of Preservation by an Enclose-and-Deposit Method", 2005, <http://www.iwaw.net/05/papers/iwaw05-tabata.pdf>.

[GRABIT] NDIIPP/CDL, , "The GrabIt File Exchange Protocol", 2008, <http://www.escholarship.org/uc/item/8t2639xb>.

[SWORD] UKOLN/JISC CETIS, , "Simple Web-service Offering Repository Deposit (SWORD)", 2008, <http://www.ukoln.ac.uk/repositories/digirep/index/SWORD>.

Authors' Addresses

John A. Kunze
California Digital Library
415 20th St, 4th Floor
Oakland, CA 94612
US

Email: jak@ucop.edu

Justin Littman
George Washington University Libraries
2130 H Street, NW
Washington, DC 20052
USA

Email: justinlittman@gmail.com

Liz Madden
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
USA

Email: emad@loc.gov

Ed Summers
University of Maryland
0301 Hornbake Library
College Park, MD 20742-7011
USA

Email: ehs@pobox.com

Andy Boyko
1538 Winding Way
Belmont, CA 94002
USA

Email: andrew@boyko.net

Brian Vargas
1354 Quincy St. NW
Washington, DC 20011
USA

Email: brian@ardvaark.net